LA-UR-18-29012
Approved for public release; distribution is unlimited.

Title: Petri Nets for Adversarial Models using Monte Carlo Simulation

Author(s): Clegg, Benjamin Wyatt
Collins, David H. Jr.
Huzurbazar, Aparna V.

Intended for: Report

Issued: 2018-09-21

Petri Nets for Adversarial Models using Monte Carlo Simulation

B. Wyatt Clegg
Department of Statistics, Brigham Young University
David H. Collins
Aparna V. Huzurbazar
Statistical Sciences Group, Los Alamos National Laboratory

Abstract

We describe an object-oriented framework and user interface for developing models of adversarial scenarios, implemented as stochastic Petri nets. The framework is implemented using the statistical computing language R, and provides facilities for eliciting models from subject matter experts, displaying models graphically, simulating scenario execution, and presenting scenario outcomes and statistics.

1 Introduction

Adversarial scenarios, such as attacks on guarded facilities, are of great interest to the defense and intelligence communities. We describe a tool, based on stochastic Petri nets, that facilitates conceptualization of these scenarios and evaluation of alternatives for defending against attacks.

Adversarial situations typically involve multiple autonomous actors operating concurrently and interactively. These are difficult to model using tools such as Markov processes, game theory, etc. Petri nets (Reisig (1982)), originally developed to model concurrency in computer architectures, offer a powerful graphic tool for eliciting such scenarios from experts, as well as a formalism that allows inference about scenario outcomes using analytic or simulation methods. For details on adversarial scenarios and a comparison of modeling methods, including Petri nets, see Collins and Huzurbazar (2015).

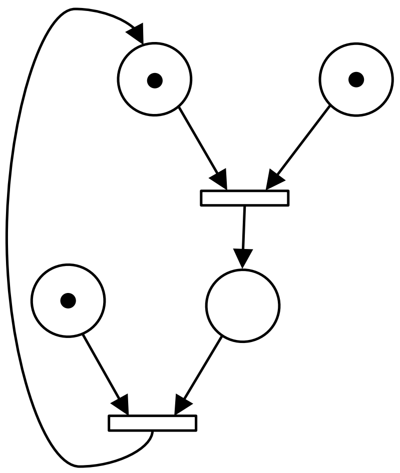


Figure 1: Graphical representation of a Petri net model.

We have implemented an object-oriented framework and user interface for developing models of adversarial scenarios, simulating scenario execution, and presenting scenario outcomes and statistics. The framework is based on reference classes in the R statistical computing language (R Development Core Team (2008)). Reference classes provide a fully object-oriented capability, including object encapsulation, methods, classes, and inheritance (Chambers (2014)). The framework also includes a graphical user interface (GUI) built using the R *shiny* package (Beeley (2016)), for displaying models and simulation outputs graphically.

2 Implementation

Figure 1 shows a graphical representation of a simple Petri net. The net is composed of *places*, represented as circles, *transitions*, represented as rectangles, *tokens*, represented as dots, and *arcs* (arrows) connecting places and transitions. Activities in a scenario are represented by the flow of tokens through the net. For further details see, e.g., Reisig (1982, 1992). Figure 2 is a diagram of the corresponding object structure within a program built using the framework. The correspondence of objects in the model with objects in the program greatly facilitates model development, graphical displays, and simulation.

Figure 3 shows the graphical display of a net in the user interface to the framework. In this case, the net was loaded from a template library,

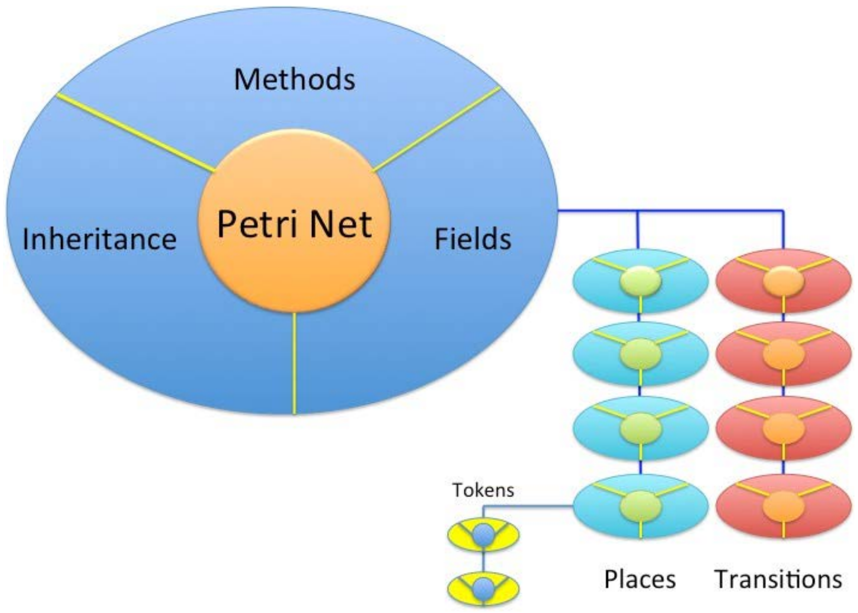


Figure 2: Object structure in the implementation of a Petri net model.

which allows reuse of common scenarios. Scenarios can be modified, or built from scratch, by adding places, transitions, and arcs. When a place is added, the initial count of tokens in the place is also specified. After running a simulation of the scenario, with parameters specified by the user, the output is shown in Figure 4.

Object classes used in the implementation include *PetriNet*, *Place*, *Transition*, *Token*, *Arc*, and specialized subclasses of these such as *Source* (a subclass of *Transition*) and *Sink* (a subclass of *Place*). Additional subclasses can be created to implement specialized behavior, but these require programming and cannot be directly created by users of the GUI.

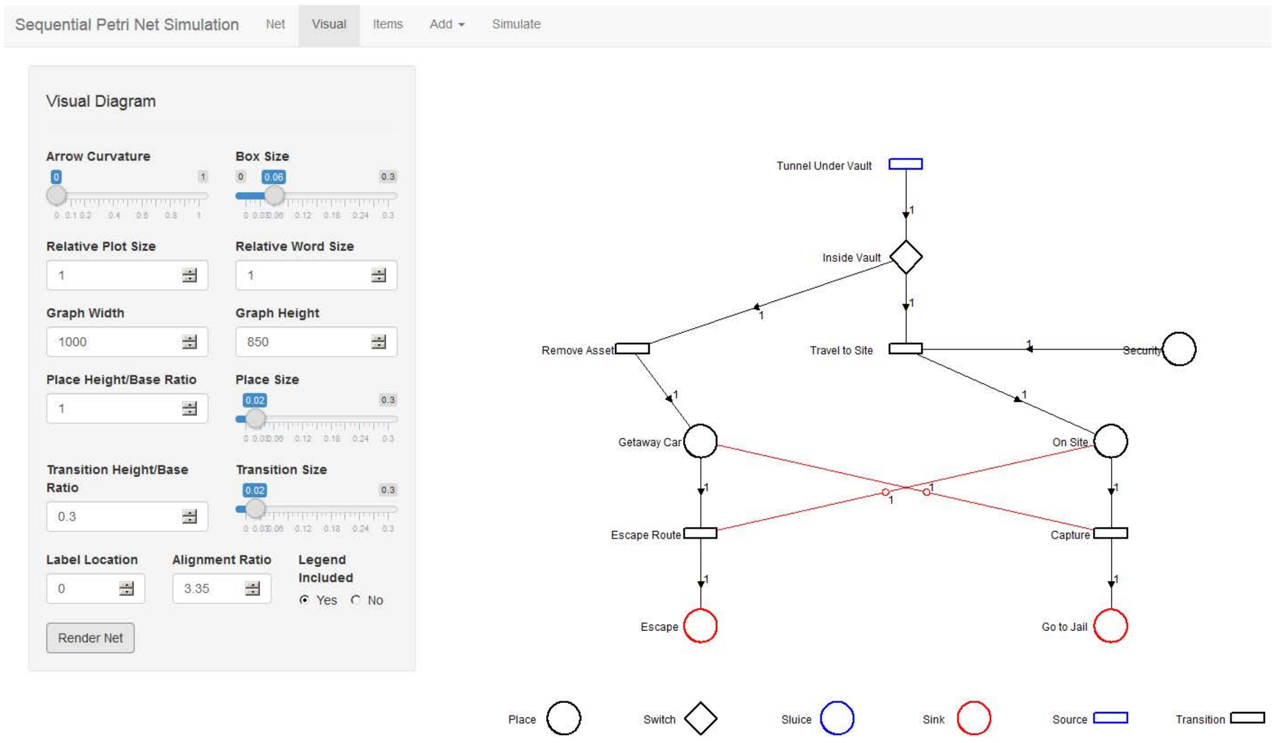
In the remainder of this paper, methods[1] implemented in a given class are designated as *ClassName::MethodName*.

2.1 Simulation algorithm

This is a general overview of the process employed to simulate iterations of a PetriNet object. The following PetriNet object methods will be described:

- *PetriNet::Branches*

[1]For those familiar with C++, methods there are called member functions

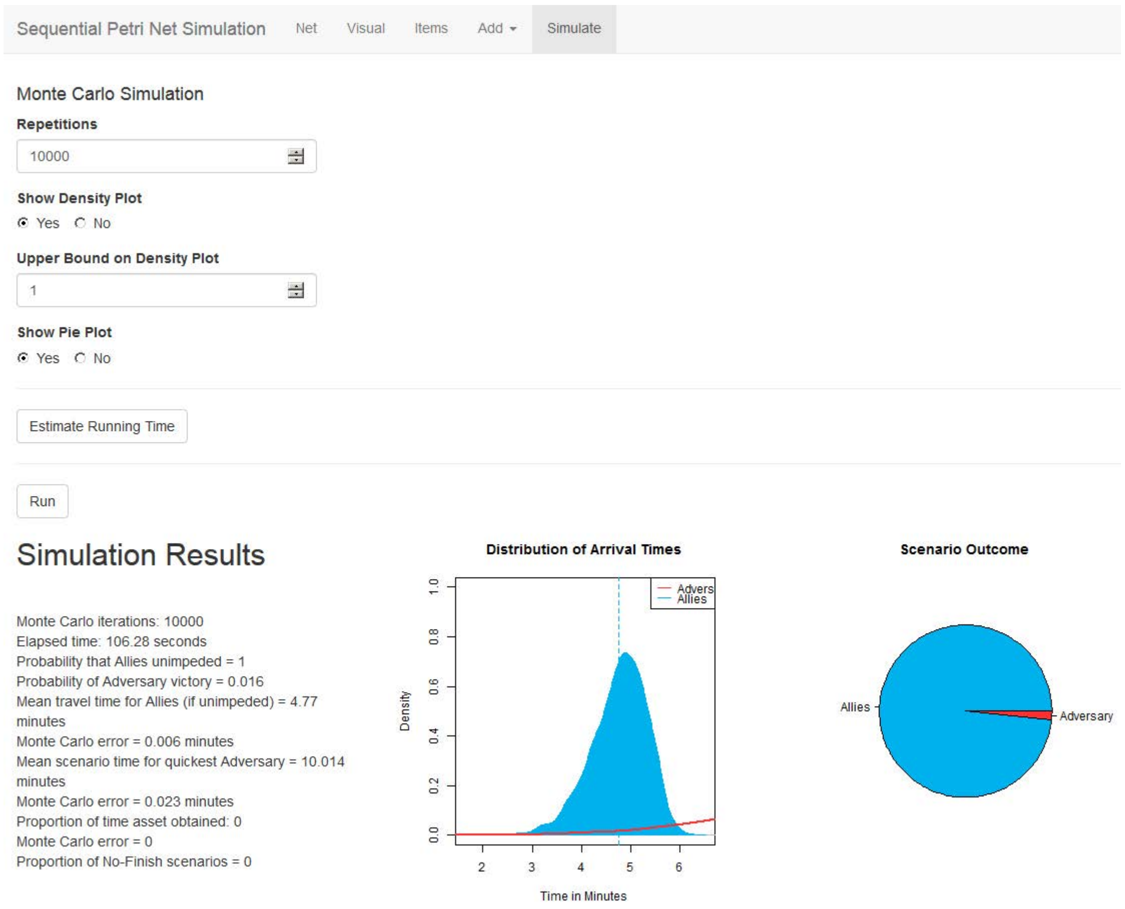


Visual Diagram

Arrow Curvature: 0
Box Size: 0.06
Relative Plot Size: 1
Relative Word Size: 1
Graph Width: 1000
Graph Height: 850
Place Height/Base Ratio: 1
Place Size: 0.02
Transition Height/Base Ratio: 0.3
Transition Size: 0.02
Label Location: 0
Alignment Ratio: 3.35
Legend Included: [✓ Yes] No

Render Net

Figure 3: Graphical display of a Petri net model. The panel on the left allows adjustment of the graphics.



Monte Carlo Simulation

Repetitions

10000

Show Density Plot

[✓ Yes] No

Upper Bound on Density Plot

1

Show Pie Plot

[✓ Yes] No

Simulation Results

```
Monte Carlo iterations: 10000
Elapsed time: 106.28 seconds
Probability that Allies unimpeded = 1
Probability of Adversary victory = 0.016
Mean travel time for Allies (if unimpeded) = 4.77
minutes
Monte Carlo error = 0.006 minutes
Mean scenario time for quickest Adversary = 10.014
minutes
Monte Carlo error = 0.023 minutes
Proportion of time asset obtained: 0
Monte Carlo error = 0
Proportion of No-Finish scenarios = 0
```

Figure 4: Output from simulation of the Petri net.

- *PetriNet::FireNet*
- *PetriNet::PrintFinish*
- *PetriNet::Iterate*
- *PetriNet::ClearNet*
- *PetriNet::GetOutArcsFrom*
- *PetriNet::InhibitableSweep*
- *PetriNet::InhibitSweep*

We will also discuss the following subclasses of *Place*:

- *Switch* objects inherit all the attributes of *Place* objects, but *Switch* objects allow for multiple arcs to exit. Furthermore, *Switch* objects will ensure that each *Arc* instance only takes the *Token* objects demanded by the multiplicity of the *Arc*. Otherwise, all *Token* objects will be passed along a functioning *Arc* out of a *Place* object.

- *Sluice* inherits the attributes of *Place*; places should be designated as *Sluice* objects when more than one arc exits or enters a place but only one arc will fire at a time. *Sluice* implements the *InhibitSweep* and *InhibitableSweep* methods so pathways will be properly fired.

- *Sink* objects inherit all attributes of *Place* objects. No arcs exit a Sink object. Furthermore, each *Sink* object has an additional field, *isWinner*, indicating whether the sink corresponds to the desired result of the simulation.

2.2 Execution algorithm

Each iteration is contained within the method *PetriNet::Iterate*. *Iterate* begins by sweeping through the Petri net using *PetriNet::InhibitableSweep*. After *InhibitableSweep* performs its function, all places where token objects may end up due to *Inhibitor* action are labeled. This label is reflected in the *IsWait* field within a place object.

PetriNet::FireNet is next employed. Within *FireNet*, the algorithm seeks out a *Transition* object of the type *Source*. Once identified, the *Source::Generate* method is called to generate the initial tokens for a simulation. These tokens are then passed to the first downstream *Place* object.

Once the tokens have been passed, the *name* field of the *Place* object is passed to *PetriNet::Branches*. *Branches* then starts at the place specified and passes tokens according to the specifications of the *PetriNet* object. *Branches* employs recursion, which explores and passes tokens down every possible path downstream of the original *Place* referenced when the function was called.

PetriNet::Branches will stop evaluating once it reaches a place labeled by *InhibitableSweep*. Tokens should stop at Places where they must be compared to determine the action of *Inhibitor* objects. For example, in a race of any kind, times of contestants are compared at the finish line, the results of that comparison determining which contestant wins. After *Branches* runs once, a similar comparison is made. *PetriNet::InhibitSweep* evaluates every inhibitor arc in the Petri net, and inhibits transitions according to the results of comparing tokens.

Note that during this step, the differences between *Switch*, *Sluice*, and *Sink* place types are incorporated. Specific references within the code all pertain to the desired attributes mentioned above.

When the simulation is complete, the *PetriNet* object is evaluated by the method *PrintFinish*, which returns a vector of length 4. Items one and three report the times of the fastest favored actor and fastest adversarial actor, respectively. Item two is a Boolean indicator, reporting whether the favored force was fastest. Item four indicates whether or not an asset was obtained by the favored actor.

Error messages will also be reported in this output. If a force was inhibited from progressing forward, their time will be reported as infinity. Furthermore, if both forces have infinite times, the second item will report a negative 1. However, despite these errors the result will still be reported.

PetriNet::Iterate then uses the method *ClearNet* to reset the *PetriNet* object to its initial state. Note that this may not correspond exactly to the state before running *Iterate* if changes have been made to the *PetriNet* after the original *Place*, *Transition*, *Arc*, and *Inhibitor* objects were created within the *PetriNet* object. *ClearNet* will work within the constraints of the GUI created for running PetriNet simulations.

Iterate then completes its run by returning the vector of four elements mentioned above. *Iterate* is used in simulate*.R scripts to create a results matrix from which summary statistics can be obtained.

References

C. Beeley. *Web Application Development with R Using Shiny.* Packt Publishing Ltd., Birmingham, UK, second edition, 2016.

J. M. Chambers. Object-oriented programming, functional programming and R. *Statistical Science*, 29(2):167–180, 2014.

D. H. Collins and A. V. Huzurbazar. Petri net models of adversarial scenarios in safety and security. Technical Report LA-UR-15-24012, Los Alamos National Laboratory, Los Alamos, NM, 2015.

R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2008. URL `http://www.R-project.org`.

W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag, Heidelberg, 1982.

W. Reisig. *A Primer in Petri Net Design.* Springer-Verlag, Heidelberg, 1992.